XML as a Model-View-Controller System for Documents

by Matthew Strawbridge

Models and Views

The Model-View-Controller (MVC) paradigm is well known by programmers as a way of separating the logical internals of a software system (the **model**) from the code concerned with presenting information to the user (the **view**). Any framework that co-ordinates the interaction between models and views is termed a **controller**. This scheme has been adopted by most modern development frameworks, since it helps software to grow over time in a flexible way, and helps to encapsulate changes. You can add new views to existing models without having to change the models themselves, and business logic can be modified without you needing to change the way it is presented to the user.

Many programmers would rightly condemn code that comprised a mish-mash of logic and presentation, but they are content to produce and consume documents that do precisely this. Memos, technical notes, meeting minutes: these are the bread and butter of the professional Software Engineer, and yet most documents are simply dumped into a word processor and left to stagnate. In this article I will describe how an MVC approach to the generation of documents can yield the same benefits that are traditionally seen with this approach to software design, and will introduce some XML [1][2] tools that can support this method. Finally, I will look at some of the alternatives to XML that could achieve the same separation of concerns.

The Problem

You may think that MVC is overkill for documents – after all, a memo is simply text; there is only one view, and that's the document you're looking at. However, what if you want to put a copy on your company intranet? I daresay your word processor has a 'save as HTML' facility. Good. What if you want to make all document references into hyperlinks; or to change the copyright text in a number of documents you've already saved as HTML; or to radically change the style of every memo. All less good.

As an example, let's take a simple document type with which we're all familiar: an ACCU book review. We already know about two views that exist on these documents: the magazine text (lets assume it's Rich Text Format), and the online review on the ACCU Web site (in HTML). Remember that, as well as having two different formats, the reviews can also have different content, since some reviews have a short version published in C-Vu and an extended version on the Web site.

XML Solution

Model

The starting point for our XML solution is to develop a Document Type Definition that describes the format of the raw information from which we will generate our documents. Strictly speaking, we could bypass this step, but then we would have no way of validating the input document – we would just try to process whatever was given. Note also that XML Schema [3][4] could have been used to provide a more detailed and robust way of validating input documents.

```
<!ELEMENT bookreview (bookdetails, reviewdetails, reviewbody)>

<!ELEMENT bookdetails EMPTY>

<!ATTLIST bookdetails title          CDATA #REQUIRED
                      author         CDATA #REQUIRED
                      isbn           CDATA #REQUIRED
                      publisher      CDATA #REQUIRED
                      pages          CDATA #REQUIRED
                      priceinpounds  CDATA #REQUIRED
                      priceindollars CDATA #REQUIRED>

<!ELEMENT reviewdetails EMPTY>

<!ATTLIST reviewdetails date     CDATA #REQUIRED
                        reviewer CDATA #REQUIRED>


<!ELEMENT reviewbody (para+)>

<!ELEMENT para (#PCDATA)>

<!ATTLIST para filter (shortonly | longonly) #IMPLIED>
```

This simple DTD just says which XML elements are allowed in a book review, and which attributes each of them may contain. Here is an example review adhering to this DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BookReview SYSTEM "BookReview.dtd">
<bookreview>
   <bookdetails title="How to Write a Book Review"
                author="B. Worm"
                isbn="0-123-45678-9"
                publisher="A. B. Cee Ltd."
                pages="123"
                priceinpounds="12.50"
                priceindollars="16.00"/>
   <reviewdetails date="2003-11-19"
                  reviewer="Matthew Strawbridge"/>
   <reviewbody>
      <para>This is an excellent book that tells you all about how to review books.</para>
      <para filter="longonly">You should buy this book because..., and finally because it's two
inches thick so it must be good.</para>
      <para filter="shortonly">Buy this book.</para>
```

```
   </reviewbody>
</bookreview>
```

This example is a hypothetical review of the book *How to Write a Book Review* by B. Worm, which was supposedly reviewed by me on the 19th November 2003. Note that the final two paragraphs provide a long description for the Web and a short description for the magazine respectively.

Views

From this single source, we want to generate the following two documents. Don't worry if you're not familiar with RTF or XHTML – the precise content that gets generated in each case is not that important; the point is that radically different target documents need to be generated from a single source.

RFT for Print

```
{\rtf

{\b How to Write a Book Review}
\par By B. Worm
\par A. B. Cee Ltd. ISBN: 0-123-45678-9,  123pp, UKP 12.50 [$16.00 (1.28) ]
\par

\par Reviewed by Matthew Strawbridge on 2003-11-19
\par This is an excellent book that tells you all about how to review books.
\par Buy this book.}
```

XHTML for Web

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1/DTD/transitional.dtd">
<html><head><title>
               Book Review -
               How to Write a Book Review</title></head><body>
   <p><em>How to Write a Book Review</em></p><p>By B. Worm</p><p>A. B. Cee Ltd.
         ISBN: 0-123-45678-9,
         123pp,
         UKP 12.50
         [$16.00 (1.28)]
      </p><hr/>
   <p>Reviewed by Matthew Strawbridge on 2003-11-19</p><hr/>
   <p>This is an excellent book that tells you all about how to review books.</p>
   <p>You should buy this book because..., and finally because it's two inches thick so it must
be good.</p>
</body></html>
```

Controllers

The main benefit of using XML to capture the model is the ease with which it can be parsed, and reshaped into different formats. This is done using XSLT [5][6], the Extensible Stylesheet Language for Transformations. An XSLT stylesheet is an XML document that uses pattern matching rules to transform an XML base document into some other form. Here are the XSLT stylesheets required for transforming our Bookreview base document into the two output types.

XSLT for Converting Bookreview to RTF

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

   <xsl:output method="text"/>

   <!-- RTF should not have any unintentional blanks, so strip them -->
   <xsl:strip-space elements="*"/>

   <!-- Template that matches the outer bookreview element
        and constructs an RTF document from it -->
   <xsl:template match="bookreview">{\rtf<xsl:apply-templates/>}</xsl:template>

   <xsl:template match="bookdetails">
<!-- Make the title bold -->
{\b <xsl:value-of select="@title"/>}
\par By <xsl:value-of select="@author"/>
\par <xsl:value-of select="@publisher"/> ISBN: <xsl:value-of select="@isbn"/>,  <xsl:value-of
select="@pages"/>pp, UKP <xsl:value-of select="@priceinpounds"/> [$<xsl:value-of
select="@priceindollars"/> (<xsl:value-of select="@priceindollars div @priceinpounds"/>) ]
\par
   </xsl:template>

   <xsl:template match="reviewdetails">
\par Reviewed by <xsl:value-of select="@reviewer"/> on <xsl:value-of select="@date"/>
   </xsl:template>

   <xsl:template match="para">
      <!-- Include paragraphs only if they either have no filter, or
           if the filter is set to 'shortonly' -->
```

```
      <xsl:if test="not(@filter) or @filter='shortonly'">
\par <xsl:value-of select="."/>
      </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

XSLT for Converting Bookreview to XHTML

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

   <xsl:output method="xml"
               doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
               doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1/DTD/transitional.dtd"
               encoding="ISO-8859-1"
               indent="no"/>

   <!-- Template that matches the outer bookreview element
        and constructs an XHTML page from it -->
   <xsl:template match="bookreview">

      <html>
         <head>
            <title>
               Book Review -
               <xsl:value-of select="bookdetails/@title"/>
            </title>
         </head>
         <body>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="bookdetails">
      <p><em><xsl:value-of select="@title"/></em></p>
      <p>By <xsl:value-of select="@author"/></p>
      <p>
         <xsl:value-of select="@publisher"/>
         ISBN: <xsl:value-of select="@isbn"/>,
         <xsl:value-of select="@pages"/>pp,
         UKP <xsl:value-of select="@priceinpounds"/>
         [$<xsl:value-of select="@priceindollars"/>
          (<xsl:value-of select="@priceindollars div @priceinpounds"/>)
         ]
      </p>
      <hr/>
   </xsl:template>

   <xsl:template match="reviewdetails">
      <p>Reviewed by <xsl:value-of select="@reviewer"/>
         on <xsl:value-of select="@date"/></p>
      <hr/>
   </xsl:template>

   <xsl:template match="para">
      <!-- Include paragraphs only if they either have no filter, or
           if the filter is set to 'longonly' -->
      <xsl:if test="not(@filter) or @filter='longonly'">
         <p>
            <xsl:value-of select="."/>
         </p>
      </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

Performing the Transformation

To actually apply these rules to the base document, an XSLT processor must be used. There are a number of these available as open source projects on the Web, the two most well-known being Xalan [7] and Saxon [8].

These templates really specify only the minimum amount of information that is needed to generate the final documents – there is very little wasted effort. Imagine deciding instead to solve the program 'programatically', and parse and transform the XML in a C++ or Java program. Indeed, XSLT is a lot more powerful than these simple examples show.

Further Improvement

These files are only meant as a demonstration of the method, and there are many improvements that could be made to create a better system for real-world use:

- **priceindollars** should be **#IMPLIED** (meaning optional), rather than **#REQUIRED**. In fact, the system should be changed to cope with various types of currency.

- It would be useful to add an enumerated summary, with options such as 'highly recommended', 'recommended', 'not recommended' and 'recommended with reservations'.
- Typically, you would want to batch process a number of reviews at once (or even every existing review, in the case of applying an updated template to the Web site). There should probably be an outer wrapper, such as **`<reviewset>`**, which can contain one or more **`<review>`** elements, and the XSLTs should handle generating either one long document, or a separate document for each review, from this collection of reviews. An alternative would be to set up a make file, perhaps using Ant [9], which includes support for XSLT transformations.
- The dates need translating from **`YYYY-MM-DD`** to a format that is more pleasant to read.
- In many cases it may be better to write a single XSLT that will convert from your bespoke format into Docbook, for which there are already some comprehensive stylesheets for conversion into many formats including HTML and PDF.

As it is customary to say is such situations, these improvements are left as an exercise for the reader.

Alternatives

Word Processor Styles

Most modern word processors support *styles*, whereby a set of properties can be assigned to segments of text. These styles can then be updated, and the updates will be automatically applied to all text having that style. While this follows the 'separation of concerns' regarding content and presentation, there are several key areas in which the XML method is to be preferred. The main difference is that styles do not transform the contents of the document, so our example using **`longonly`** and **`shortonly`** attributes for paragraphs could not be implemented without the use of macros. It would also be difficult to regenerate a batch of documents if the template changes, especially if multiple *Save As* formats were needed.

Microsoft Word 2003

I have read about the XML support in Microsoft Office 2003, but haven't used it myself. I would be interested to know if anyone uses it in a similar fashion to that described here – at the very least it promises to be a more user-friendly way of populating the raw XML files than simply using a text editor.

Final Word

Many software developers have a real loathing for any form of documentation. ACCU members, generally being a well-read bunch, may have less of an aversion, but one thing is clear – if you dislike writing documents, then you'll really hate having to make minor updates to several hundred of them by hand. By separating out the content (model) from the presentation (view), and creating reusable templates to generate one from the other, maintenance of an archive of documents is greatly simplified. XML and XSLT can be used to implement such an MVC treatment of documents.

As with most things in the world of computers, there is more than one way to string a cat (or should that be to **`cat.toString()`**?) I am not advocating that all documents should be written in this way, but for cases where you have lots of similar documents that do, or may, need to be rendered in more than one format, this technique should save a lot of time in the long-run at the expense of a little work up-front. Now, where have I heard that before?

References

[1] *XML*. http://www.w3.org/XML/
[2] Elliotte Rusty Harold, W. Scott Means. *XML in a Nutshell*, O'Reilly, 2002
[3] *XML Schema*. http://www.w3.org/XML/Schema/
[4] Eric van der Vlist. *XML Schema*, O'Reilly, 2002
[5] *XSLT*. http://www.w3.org/Style/XSL/
[6] Doug Tidwell. *XSLT*, O'Reilly, 2001
[7] *Xalan*. http://xml.apache.org/xalan/
[8] *Saxon*. http://saxon.sourceforge.net
[9] *Ant*. http://jakarta.apache.org/ant/index.html